
title: Send Transactions description: Step-by-step walkthrough of how to send transactions on SUAVE keywords: - practical - suave - transaction - example

:::info

Sending transactions requires that you use either the Golang or TypeScript SDKs we have made available. Confidential Compute Requests (CCRs) are not EIP-1193 compatible so signing CCRs from browser wallets requires some workarounds.

:::

The [previous tutorial illustrated](#) how to deploy a series of example contracts using the Golang SDK and, in the first tutorial, you [setup SUAVE and then ran a script to "test the devnet"](#).

This tutorial will reiterate how contract deployment works, **and show you how to craft the transactions** that occur in [this script](#).

MEV-Share Example

The suave-geth repo has an [example version of MEV-Share](#). The script we'll be walking through does four things:

1. deploy the MEV-Share contract
2. sign and submit a transaction to be back run
3. grab the extracted hint from onchain
4. sign a back run transaction and submit it

Step-by-Step Breakdown

Let's break down what this script is doing.

Step 1: Create and Fund Test Accounts

```
```go testAddr1 = generatePrivKey() testAddr2 = generatePrivKey()

if err := fundAccount(mevmClt, testAddr1.Address(), fundBalance); err != nil { return err } fmt.Printf("- Funded test account: %s (%s)\n",
testAddr1.Address().Hex(), fundBalance.String()) ```
```

- Generate two new private keys representing test accounts.
- Fund these test accounts with a specified amount of Ether to simulate transactions.

### Step 2: Deploy MEV-Share Contract

```
go // Code snippet related to the step txnResult, err := sdk.DeployContract(mevShareArtifact.Code, mevmClt)
```

- Deploy the MEV-Share smart contract to the SUAVE chain using the bytecode and a client instance.
- Confirm the deployment by checking the transaction receipt status.

### Step 3: Send Confidential Compute Request

In this example, the transactions in the bundle would be from other blockchains, and may contain MEV on those chains which the user wants to share with searchers listening for hints on SUAVE.

```
```go refundPercent := 10 bundle := &types.SBundle{ Tx: types.Transactions{ethTxn1}, RevertingHashes: []common.Hash{},
RefundPercent: &refundPercent, } bundleBytes, _ := json.Marshal(bundle)

// new mevshare transaction inputs
targetBlock := uint64(1)
allowedPeekers := []common.Address{mevShareContract.Address()}

confidentialDataBytes, _ := bundleContract.Abi.Methods["fetchConfidentialBundleData"].Outputs.Pack(bundleBytes)

txnResult, err := mevShareContract.SendTransaction("newTransaction", []interface{}{targetBlock + 1, allowedPeekers, []common.Address{}}, confidentialDataBytes)
if err != nil {
    return err
}
```
```

- Craft a Confidential Compute Request using the MEV-Share contract function `newTransaction`.
- Serialize the bundle of transactions to include in the CCR, along with the `refundPercent` the sender of the CCR is looking for.
- Sets a `targetBlock` which is used as the `decryptionCondition`, along with the `allowedPeekers`. That is, the data in the bundle may be decrypted after a specific block (i.e the `targetBlock`) by any of the `allowedPeekers` (which, in this context, is the `mevShareContract` itself who needs to store the bundle and potentially match it with any backruns it receives once a hint is emitted). This is what we mean by "programmable privacy" which, in this case, is being used to ensure pre-trade privacy because the `decryptionCondition` is just a block number.
- Send the CCR to the specified Kettle and validate its inclusion.

### Step 4: Send Backrun

In practice, this transaction would be sent by a searcher listening for hints on SUAVE, after they have received the hint emitted by the newBid function that the user called above, crafted a backrun based on that hint, which they want a builder to include on the original domain where the user wants to transact.

```
``go backRunBundle := &types.SBundle{ Tx: types.Transactions{ethTxnBackrun}, RevertingHashes: []common.Hash{}, }
backRunBundleBytes, _ := json.Marshal(backRunBundle)

confidentialDataMatchBytes, _ := bundleContract.Abi.Methods["fetchConfidentialBundleData"].Outputs.Pack(backRunBundleBytes)

// backrun inputs
targetBlock := uint64(1)
allowedPeekers := []common.Address{mevShareContract.Address()}

txnResult, err := mevShareContract.SendTransaction("newMatch", []interface{}{targetBlock + 1, allowedPeekers, []common.Address{}, DataID}, confidentialDataMatchBytes)
if err != nil {
 return err
}

``
```

- Create a backrun transaction bundle that will be sent after the initial bid. **This is also a Confidential Compute Request**
- Serialize the backrun bundle and use the MEV-Share contract's newMatch function to submit it.
- Verify the transaction receipt to ensure the backrun was successful.

:::info

What is not included in this script is the final step in the e2e example. We need to send this matched bundle - which includes both the original transaction from the user and the backrun from the searcher - to a builder so that it can be included in a block on the target domain. Rigil includes the ability to send such bundles to centralized relays for inclusion. There is [builder solidity code which demonstrates how this might occur here](#).

:::

## Result

If all has been successfully run you should see the following in your terminal:

```
bash suave-geth$ go run suave/devenv/cmd/main.go Step 0: Create and fund test accounts - Funded test account: 0x66d5a8D6B34329c0639071275b3d78D29e11EbC6 (1000000000) Step 1: Deploy mev-share contract - Mev share contract deployed: 0x8f21Fdd6B4f4CacD33151777A46c122797c8BF17 Step 2: Send user transaction - transaction sent at txn: 0xb49debcdead2b306d6ab6282b88fdad7c8d6a33d87df34b79f56d141eae7c08a - Data record id: 30bbc65298f24e67aaf5c95bf5f0686c Step 3: Send backrun - Backrun sent at txn: 0xc7880e61e94aaab48c60655c321716ecab6edab752586448b0412e93a969889 - Backrun data record id: db98b83d02694fc2b13c042ad22c233
```